JUnit Testing Exercises

Exercise 1: Setting Up JUnit

Scenario:

You need to set up JUnit in your Java project to start writing unit tests.

Steps:

1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

2. Add JUnit dependency to your project. If you are using Maven, add the following to your pom.xml:

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.13.2</version>
  <scope>test</scope>
</dependency>
```

3. Create a new test class in your project.

pom.xml (JUnit-basic) | Main.java | Calculator.java | AssertionsTest.java | CalculatorWithSetupTest.java

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>JUnit-basic</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency...>
    </dependencies>


</project>
```

Exercise 2: Writing Basic JUnit Tests

Scenario:

You need to write basic JUnit tests for a simple Java class.

Steps:

1. Create a new Java class with some methods to test.

2. Write JUnit tests for these methods.

```java
package com.example;

6 usages
public class Calculator {
    2 usages
    public int add(int a, int b) {
        return a + b;
    }

    2 usages
    public int subtract(int a, int b) {
        return a - b;
    }
}
```

```java
package com.example;

import junit.framework.TestCase;
import org.junit.Test;
import static org.junit.Assert.*;

public class CalculatorTest extends TestCase {
    @Test
    public void testAdd() {
        Calculator calc = new Calculator();
        assertEquals( expected: 5, calc.add( a: 2,  b: 3));
    }
    @Test
    public void testSubtract() {
        Calculator calc = new Calculator();
        assertEquals( expected: 1, calc.subtract( a: 3,  b: 2));
    }
}
```

```
CalculatorTest (com.example)   6 ms
    testAdd                    6 ms
    testSubtract               0 ms

Tests passed: 2 of 2 tests – 6 ms

"C:\Program Files\Java\jdk-21\bin\java.exe"

Process finished with exit code 0
```

Exercise 3: Assertions in JUnit

1. Write tests using various JUnit assertions.

```java
package com.example;


import static org.junit.Assert.*;
import org.junit.Test;
public class AssertionsTest {
    @Test
    public void testAssertions() {
        assertEquals( expected: 5,  actual: 2 + 3);
        assertTrue( condition: 5 > 3);
        assertFalse( condition: 5 < 3);
        assertNull( object: null);
        assertNotNull(new Object());
    }
}

```

```
AssertionsTest (com.example)  7 ms
    testAssertions            7 ms
```

```
Tests passed: 1 of 1 test – 7 ms
"C:\Program Files\Java\jdk-21\bin\java.exe"

Process finished with exit code 0
```

Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit

Scenario:

You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

Steps:

1. Write tests using the AAA pattern.

2. Use @Before and @After annotations for setup and teardown methods.

```java
package com.example;

import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.After;
import org.junit.Test;

public class CalculatorWithSetupTest {
    4 usages
    private Calculator calculator;
    @Before
    public void setUp() {
        System.out.println("Setting up test...");
        calculator = new Calculator(); // Arrange
    }
    @After
    public void tearDown() {
        System.out.println("Cleaning up after test...");
        calculator = null;
    }
    @Test
    public void testAdd() {
        int result = calculator.add( a: 4,  b: 5);
        assertEquals( expected: 9, result);
    }
    @Test
    public void testSubtract() {
        int result = calculator.subtract( a: 10,  b: 4);
        assertEquals( expected: 6, result);
    }
}
```

✓ CalculatorWithSetupTest (com	15 ms
✓ testAdd	14 ms
✓ testSubtract	1 ms

✓ Tests passed: 2 of 2 tests – 15 ms

```
"C:\Program Files\Java\jdk-21\bin\java.exe"
Setting up test...
Cleaning up after test...
Setting up test...
Cleaning up after test...
```